Universidade Federal de Uberlândia

Isadora Rezende Lopes e Lucas Tannús Vieira

Atividade Prática - Análise de Algoritmos

Uberlândia, Brasil
2020

1 Equipe

Nome: Isadora Rezende Lopes. Matrícula: 11611BCC051.
Nome: Lucas Tannús Vieira. Matrícula 11611BCC049.

2 Descrição dos Experimentos

2.1 Linguagem de Programação

A linguagem de programação adotada para o desenvolvimento dos algoritmos propostos na atividade prática de Análise de Algoritmos foi Python. Os algoritmos foram executados utilizando a versão 3.8.6 do python. As bibliotecas NumPy e Math foram utilizadas na construção de matrizes e funções matemáticas. A biblioteca Time foi utilizada para computar o tempo de execução dos algoritmos.

2.2 Matrizes

Inicialmente, a proposta desta atividade prática era gerar matrizes quadradas com valores de n de 500 até 1000, incrementando o valor de n de 50 em 50 (exemplo: 500, 550, 600, 650 ... 950, 1000). Uma rotina utilizando a função np.random.randint da biblioteca Numpy foi desenvolvida para gerar duas matrizes com cada uma das proporções nxn, com valores aleatórios de 1 até 5000. Dessa forma, 10 pares de matrizes foram gerados.

Após executar os experimentos para os valores de n estabelecidos, a fim de obter mais informações sobre o comportamento dos dois algoritmos, surgiu a necessidade de executar mais experimentos, dessa vez utilizando apenas matrizes 2^n x 2^n. Nesse caso, utilizando a mesma rotina construída para gerar as matrizes anteriores, foram geradas matrizes 2x2, 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024 e 2048x2048. Logo, mais 11 pares de matrizes foram gerados. No total, foram gerados 21 pares de matrizes para este trabalho.

2.3 Réplicas de Experimentos

Para cada par de matrizes, vinte réplicas do experimento foram executadas, a média e o desvio padrão de cada conjunto de experimentos foram retirados e utilizados para a construção de gráficos e tabelas referentes ao desempenho dos dois algoritmos. No experimento em que n era igual à 2048, apenas 5 réplicas foram executadas, devido ao alto tempo de execução.

3 Desenvolvimento

Para o experimento com matrizes de tamanhos 2^n obtivemos os seguintes resultados:

Figura 1: Tempos Algoritmo 1 para matrizes com dimensões 2^n

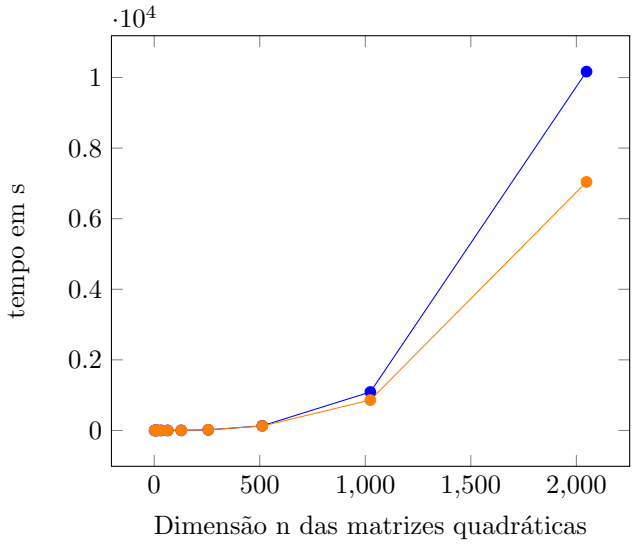
Tamanho	Tempo médio	Desvio Padrão
2	2.3e-05	3E-06
4	0.000161	0.000219
8	0.000537	2.2e-05
16	0.003766	0.000308
32	0.028527	0.001399
64	0.233136	0.014311
128	2.042591	0.0608
256	16.192066	0.10547
512	130.817115	0.637894
1024	1088.99034	3.246443
2048	10166.9074	4.585664

Figura 2: Tempos algoritmo de Strassen para matrizes com dimensões 2^n

Tamanho	Tempo médio	Desvio Padrão
2	3.5e-05	7E-06
4	0.000231	6.2e-05
8	0.001095	9.6e-05
16	0.006818	0.000402
32	0.046472	0.00203
64	0.329325	0.028087
128	2.557854	0.059935
256	17.678188	0.11212
512	124.79019	3.244603
1024	861.017619	2.644007
2048	7041.9525	3.005176

Com base nas tabelas acima é possível destacar que o Algoritmo 1 é mais eficiente que o algoritmo de Strassen para matrizes com dimensões de até 256.

Além disso, abaixo temos um gráfico que deixa mais evidente o desempenho de cada algoritmo para as dimensões propostas, exibindo a média de tempo para ambos os algoritmos. Em azul temos a média do Algoritmo 1 e em laranja os tempos para o algoritmo de Strassen.



A partir da construção do gráfico contendo as médias dos experimentos representado na Figura 1, é possível perceber que para matrizes 2^n x 2^n pequenas, os algoritmos 1 e Strassen possuem comportamentos semelhantes, ou seja, tempos de execução baixos. Para matrizes de grandes dimensões, como 512, 1024 e 2048 podemos observar que o desempenho do Strassen passa a ser cada vez mais superior ao do Algoritmo 1.

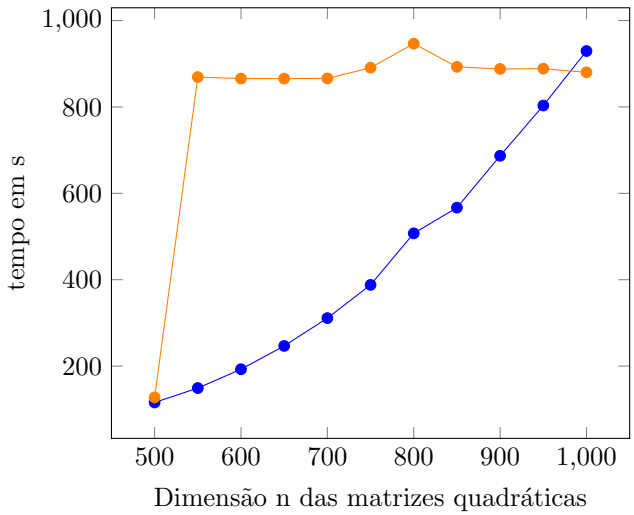
Da mesma forma foram executados experimentos para matrizes com dimensões no intervalo de 500x500 a 1000x1000 e o valor de n foi incrementado de 50 em 50. As Figuras 2 e 3 apresentam as médias dos tempos estimados.

Figura 3: Tempo médio do Algoritmo 1

Tamanho	Tempo médio
500	115.7824
550	149.0848
600	192.6684
650	246.6591
700	311.1332
750	388.0501
800	507.3416
850	566.7948
900	686.9446
950	803.3104
1000	929.5353

Figura 4: Tempo médio do Strassen

Tamanho	Tempo médio
500	127.5364
550	869.1958
600	865.7599
650	865.6560
700	866.1223
750	890.8585
800	946.5751
850	892.9137
900	888.0189
950	888.6766
1000	880.1169



Para casos em que as dimensões das matrizes são diferentes de 2^n x 2^n fica claro que o algoritmo de Strassen não é tão efetivo quanto o Algoritmo 1. A grande diferença entre as médias dos resultados dos algoritmos pode ser explicada pelo preenchimento da matriz com zeros.

É possível perceber no caso de n igual a 1000, como temos apenas 24 zeros para preencher a matriz até obter o tamanho 1024, o Strassen possui desempenho superior ao Algoritmo 1. Nos demais casos, devido ao preenchimento da matriz com muitos zeros o Strassen obtém um desempenho muito inferior ao desempenho do Algoritmo 1.

Com base nesse experimento fica evidente que o algoritmo de Strassen não é tão efetivo na multiplicação de matrizes contendo muitos zeros, uma vez que nos casos acima as matrizes foram completadas com zeros para que tivessem dimensões da forma 2^n.

4 Conclusão

Com base nos resultados obtidos neste trabalho, é possível concluir que ao multiplicar matrizes de dimensões pequenas ou matrizes contendo muitos zeros o Algoritmo1 possui um bom desempenho. Em casos que há necessidade de multiplicar matrizes de dimensões 2^n muito grandes o Strassen possui o melhor desempenho.